

THE QUALITY OF SOFTWARE IS MANAGEABLE

How do you measure and ensure the quality in software? Here's what it means for manufacturing.

BY LAWRENCE GOULD

Software can never be totally bug-free. It is commonly accepted that no software with more than 10,000 lines of code has ever formally been proved correct. Given this, one need not look at the government's Star Wars project to shudder. Bugs in manufacturing plan-

ning and control software can prove devastating in actual costs, lost production, and lost customers.

A software vendor's quality assurance can eliminate most of the killing bugs so that the user can be reasonably sure that the program will function as specified. However, users typically expect the quality of software to be better than that of any

other product they buy. They shouldn't, though.

"You have to treat software like any other manufactured product," says Lou Mazzucchelli, chief technical officer of Cadre Technologies Inc. "All the same rigor should apply. There's not a company in the world that doesn't have a well-formulated set of procedures and control mechanisms for either manufacturing or changing their product. It is virtually the same whether for metalworking or for software manufacturing."

Adds John Perkins, software programmer for Dynamics Research Corp., "The question shouldn't be whether the software is right or wrong. It should be whether the software is useful or not useful. And when it breaks, do you know it?"

Software quality is often expressed in vague terms such as reliability, maintainability, and efficiency—terms with different meanings depending on who is talking, the user or the programmer. As a result, a variety of software users, industry groups, and standards organizations are working to define those terms in precise, measurable parameters.

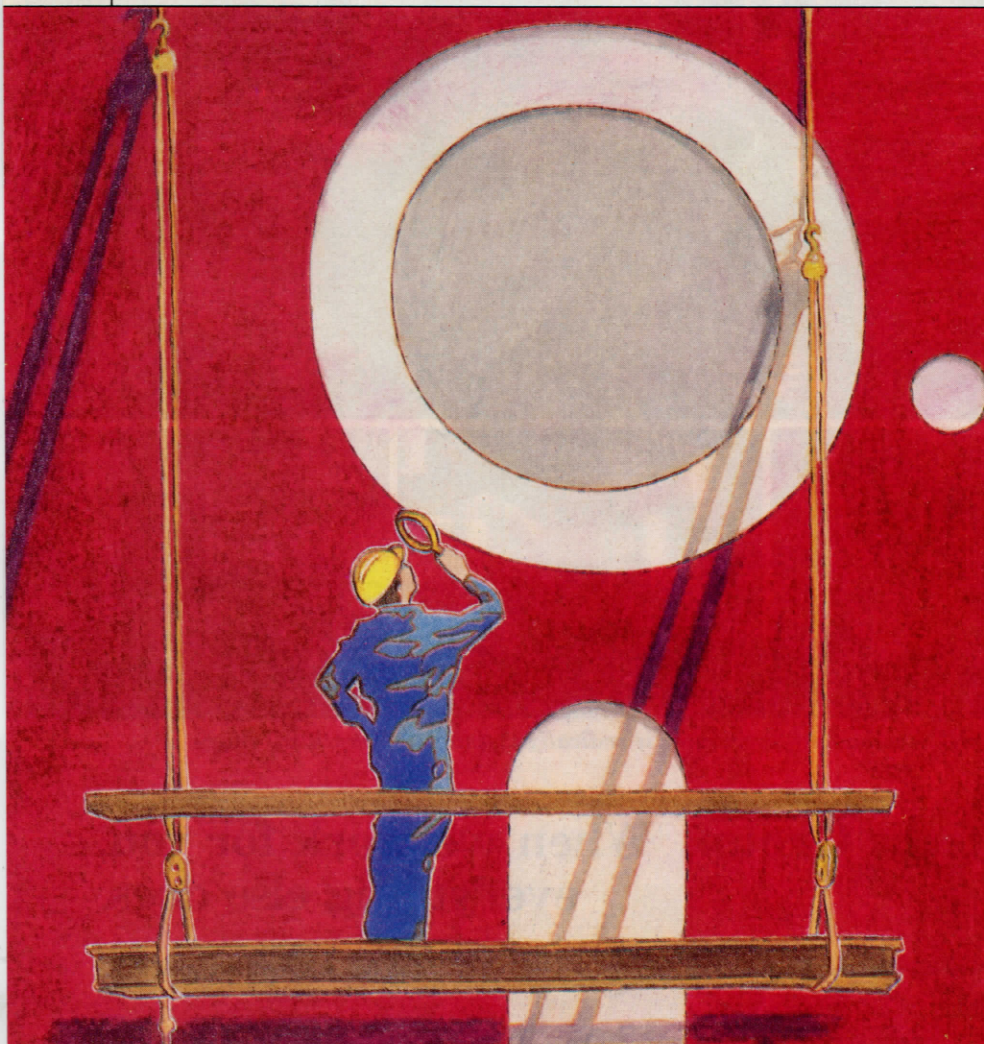
For instance, the Rome Air Development Center at the Griffiss Air Force Base has spent 10 years developing a methodology for specifying and evaluating software quality attributes. Both the Institute of Electrical and Electronics Engineers and the American Society for Quality Control have committees that develop standards on such topics ranging from software development to software quality. Some of these quality attributes are functionality, performance and efficiency, code coverage, accuracy, configuration and portability, maintainability, redundancy, and conformance to standard.

Some quality attributes are easily quantifiable. With performance, either the software does what it is supposed to do in a given amount of time or it doesn't. With adherence to standards, either the software meets a list of functions or it doesn't.

However, the metrics for some quality attributes are harder to define. One such metric is the McCabe Complexity Measure, which counts such code features as the number of branches in the software program. This measure assumes that with more branches or paths through a program, the more complex it is, and the more likely it is to have an error.

The most important software

Illustration: Bob Clark



quality metric, says Marty Browne, vice president of applications development at ASK Computer Systems Inc., "is whether that software performs the way the specification and the documentation say it is supposed to perform."

Adds Nick Stewart, software quality engineer at Rockwell International Satellite & Space Electronics Division and chairman of the Software Quality Technical Committee for the ASQC, "The definition of quality software is like any other product in the world: It's in the eyes of the beholder."

Not surprisingly, quality costs, or rather, non-conformance to software requirements costs, go up as the life cycle progresses. According to Stewart, 70% of the life-cycle cost of software is in the maintenance phase—that is, when the software is in the customer's hands. Moreover, 60% of all software errors can be traced to the requirements at the design phase.

Explains Mazzucchelli, "Programmers generally do not make a mistake translating their intent into code, and syntactic errors are easy to find. However, if the programmers are working from bad specifications, they'll do a good job of translating the bad specifications into code."

Continues Stewart, "It costs as much as six to ten times to correct an error discovered during the maintenance phase as it would if the error were caught in the design phase. In some highly complex aerospace applications, that estimate goes as high as one hundred times."

According to studies by AT&T Bell Labs, it costs nothing to fix an error in the initial design phase, \$100 to fix the same error once it is coded, and \$10,000 to fix the problem after the software is released.

Stewart explains: "The cost of an error as the software product passes into the requirements phase of development would include the time spent by those who negotiated the change; namely, the contracting department and the analyst who performed the study and made changes to the requirements documents, plus printing. These costs shouldn't be high."

"At the design phase, all outstanding issues of requirements are supposed to be resolved. Any changes from this point on will incur the additional costs of modifying a solid set of documents."

Percent of Sources of Software Development Errors

Analysis:

- Requirements Incorrect or Misinterpreted
- Functional Specification Incorrect or Misinterpreted

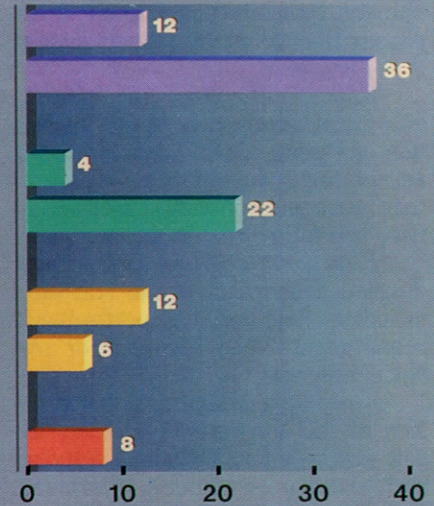
Design:

- Design Error Involving Several Components
- Error in Design or Implementation of Single Component

Implementation:

- Clerical Error
- Error Due to Previous Misconception of an Error

Others



"The cost of an error in the coding phase includes the cost of all those who participated in the analysis to resolve the error, and the time to implement the change. This cost is compounded by the number of interfaces to other modules. If an error is not a simple correction to logic or naming convention, the time and labor to correct it can be extensive. It is not uncommon to find 40% of the development cost in the test phase."

Says Ted Williams, president of

Comac Systems Corp., a producer of computer-aided maintenance management software, "Quality software is money in the bank. It just makes good sense from a business standpoint to have the highest-quality software application possible. Whatever we can do to improve that quality during the process of developing, releasing, and upgrading that software, the less time we have to spend fixing the software, repairing errors, and potentially damaging our image."

What's Bugging You?

Software errors, called bugs, come in several varieties:

Typographical and coding errors are typically transcription errors that are easily found through proofreading and automated debuggers that check each line of code for operability. A keystroke error, though, may appear as a logical error. For example, in C language, a double equal (=) sign is a test between the two objects on either side of the equal signs, but a single equal (=) sign assigns the value to right of the sign to the object on the left.

Logic errors can be as simple as a mindless mistake, such as not writing what was intended or using one identifier for another, or they can be the result of a programmer losing the thread of a complex analysis translated into code.

Project management errors occur because one programmer assumed that another programmer was handling an aspect of the software project. These errors include missing code critical for the software to operate, missing functionality, multiple labels for the same data elements, and unlinkable software modules.

Interface errors occur when two or more software modules within the same program cannot exchange data because data elements are not defined properly, are mislabeled, or are in some other way insufficient. Interface errors are often the result of inadequate project management, but they can also be design errors, especially in revised or added modules to an existing software program.

Misinterpretation of specifications are not necessarily errors in software code. Explains Max Hitchens of HEI Corp., "In this case, the software does what I tell it to do, not what I want it to do."

"Does the program perform the way it was designed to perform or perform in a way that solves the customer's problem?" says Marty Browne of ASK Computer Systems. "Unfortunately, these may be two different issues."

Explains Fabrizio, "System malfunctions resulting from misinterpreted specifications are perceived as software errors by the customer, but the system may be doing exactly what the developer had intended! In this case, the developer and the customer have interpreted the specification differently."

Source: AT&T Bell Labs

Chart: PIX•Elation/Fran Heyl Associates, NYC

To combat costs, software must be developed and tested according to rigorous standards. Software development typically consists of seven phases that generally follows design, code, and test:

Conceptualization: the customer's initial description of the application, including needs, hardware, interrelated automated systems (hardware and software), user expertise, and future requirements.

Functional requirements: a more detailed description of the software, including functionality, performance requirements, and the user or machine interface.

Functional design: a detailed design of the software created by the software vendor, including software architecture, modules, interfaces, and data requirements (inputs, outputs, and how they are processed).

Implementation: writing the software code and initial debugging.

Testing: checking out the software for logic, functionality, inconsistencies, unresolved linking, and other errors through a variety of testers, including static testers, regression testing, simulation, and emulation.

Installation and checkout: a full-functional test of the software in its operational environment.

Maintenance: modifications to the released software to correct software errors, revise or add functionality, or port to other systems.

Software has to steep like good wine," explains Williams. "It takes a number of iterations to remove software inconsistencies and bugs. And there's no shortcut to doing that; software testing takes time and effort."

Robert Fabrizio, director of factory systems, ITP Boston Inc., agrees. "There seems to be a strong indication that the longer you spend

in design, the less time is required in coding and debugging a piece of software. It's a trade-off between design time and debugging."

However, there is a catch to testing software: How do you know you're done? "There is no good answer to that question," replies Fabrizio. "As a system becomes more complex, it's generally understood that you can never test every possible condition that could ever arise. However, a good measure is the time between errors. The longer you run the system to find the next error, the closer you are to the end of the test. In fact, some users will specify that acceptance tests run 40 hours, 10 days, or whatever, without an unrecoverable error."

Although no test can replace a user in an actual operating environment, software is typically used to test and ensure the quality of software. These software tools are available at both the front and back ends of software development. For instance at the front end, computer-aided software engineering (CASE) provides software design with what computer-aided engineering provides hardware design.

CASE typically allows programmers to write, compile, test, and debug code at the same terminal. It attempts to put as much of the development effort up front in the software design cycle—namely, during the functional requirements and design phases—thereby reducing the errors (and costs) that result from faulty detailed design specification.

One CASE product—Teamwork from Cadre Technologies—provides real-time modeling, systems analysis, system simulation, source code generation, language tools, debuggers, software performance analysis, and verification. Teamwork allows programmers to hypothesize software designs and simulate performance before investing time in development.

Other Teamwork modules, from Cadre's MicroCASE Division, provide back-end support by emulating off-the-shelf microprocessors and software analyzers, thereby helping programmers observe code execution and performance.

Another approach to back-end performance testing is to simulate or emulate the manufacturing equipment and then run the target software as if it was in an actual operation. Explains Max Hitchens, president of HEI Corp., a supplier of real-time emula-

How Does Software Measure Up?

Some of the quality attributes of software are as follows:

Functionality measures whether the software does what it is supposed to do. Functionality must be measured dynamically: When the software is being used and being compared to the initial customer specifications.

Performance and efficiency measures whether the software is fast enough compared to an existing software product or previous revision, or perceived as fast by the user. For example: Does the software keep up with the hardware—that is does the disk access speed slow the software? Does it keep up with the operator—that is, does the cursor move with the mouse controller? Does it keep up with the application—that is, can the software collect the data when needed?

Code coverage measures whether all of the code in a program is being used. Excess code can affect a program's performance, efficiency, and reliability.

Accuracy measures whether the software counted the data correctly and whether it evaluated an action, especially an emergency action, properly. To a large extent, a test of the software's accuracy is really a test of its functionality.

Configuration measures whether the software operates as required on the platform specified, including hardware and its operating system, display monitor, peripherals, networking system, and other applications software. Equally important, does all the software functionality work on every configuration of that hardware; for example, does the software display data on a monochrome as well as on a color screen?

Portability is another aspect to the configuration attribute; that is, can the software be used on different hardware platforms. For example, does a software application written in MS-DOS work for all PC-compatibles?

Maintainability measures the ease that software can be changed, enhanced, fixed, and updated.

Redundancy, also known as fault tolerance, resource recoverability, or error recovery, is probably one of the most critical aspects of software. This test evaluates whether the software handles spurious inputs and errors "gracefully." For example, if the operator keys in a six-digit number when the software expects a five-digit number, what should the software do?

Conformance to standards determines whether the software—either its coding structure or its functionality—adheres to a standard, if one exists. In some applications, translating a standard into software is critical. For example, aerospace and defense industries must meet stringent Department of Defense quality and traceability specifications when assembling defense systems. In accounting, there are "accepted accounting practices," which directly affect the profit and loss of a company.

In manufacturing, explains Christopher Gray, president of Gray Research, "a measure of the quality of manufacturing resource planning (MRP II) software is how well does it conform to the functions outlined in the Standard System, first offered in 1976 by Oliver Wight. Specifically, does the MRP II software include all the functionality for master production scheduling, demand management, time-phased requirements planning, sales and operations planning, and so on?"

tion systems, "A PC-based simulator can be connected to programmable controllers or computers to both test the software implementation and debug the control system.

"As an emulator, it can replace conveyors, storage systems, guided vehicles, robots, and other physical computer-based equipment so that the control system can be tested in real time. The emulation can show that the software is operational and that it meets customer functional and performance specifications before installation, eliminating expensive and time-consuming field testing."

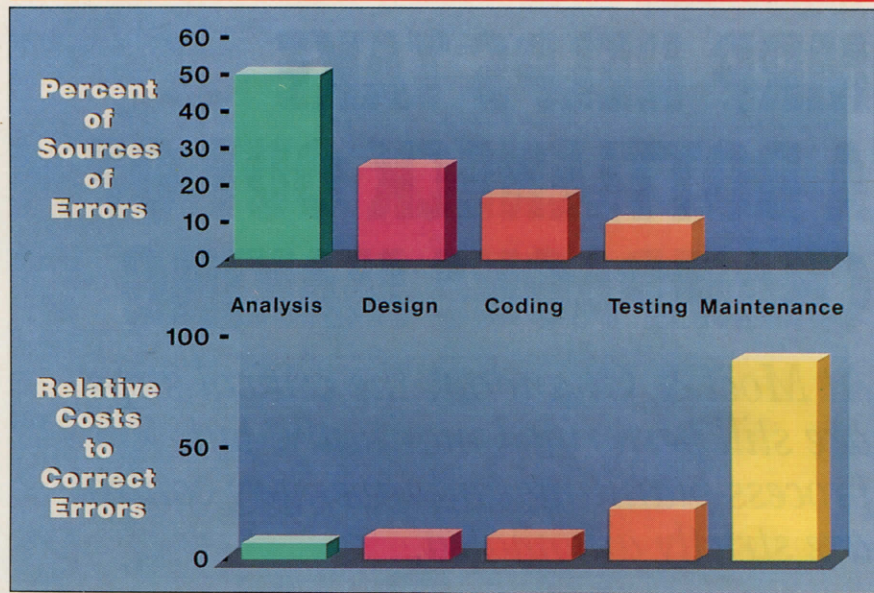
Unlike simulation and emulation, which tests software dynamically, other back-end software tools provide static tests for software quality. Adamat from Dynamics Research details whether Ada language source code reflects Department of Defense software engineering goals. The software test system checks more than 150 Ada-specific metrics to determine if the code adheres to the accepted Ada quality goals of maintainability, reliability, and portability.

A similar product, Scoreboard, from TravTech Inc., performs static code quality analysis for COBOL programs. This package rates the complexity and structure of software based on standards from independent IEEE studies and weighted by the user. The program outputs an overall score from 0 to 100 for the software package, as well as scores individual software metrics so that the user can determine which aspects of the software need to be revised.

Software is purchased *caveat emptor*: let the buyer beware. Because of that, software vendors have been accused of "getting away with murder." Some, of course, are reversing the status quo. Says Comac's Williams, "If somebody does something totally unusual and tells us, we either fix the bug in the next release or give the user a way to work around the bug, such as an instruction or a quick fix sent by modem."

Sun Microsystems Inc. (Mountain View, Calif.) uses its Network Software Environment to manage all the objects created while developing software, including data, modules, and documentation. Explains George Symons, CASE product line manager at Sun, "With operating system software, there is the issue of releasing and re-releasing software and managing multiple versions of that software for different architectures. To do

Costs to Correct Errors Increase with Each Stage



that, we must have a release mechanism to ensure the distribution of the right pieces of software and documentation are current."

But how can the software customer ensure the quality of software? "The first thing you can do to improve the quality of software," says Mazzucchelli, "is to get it right from day one, and that means you have to pay a lot of attention to the specification and detailed design of the software, which points back to knowing your business and what you want."

Explains Fabrizio, "Software development ideally should be a joint effort between the customer and the software vendor. The customer should be involved up front in the functional specification of the system. The customer should also be involved in the back end by defining the acceptance test plan, to verify whether the software meets specifications and the customer's needs, and run the test to ensure proper operation, fault disposition, and acceptance."

The customer should investigate the software itself: its age, the frequency of software updates, and the vendor's future software plans. Says Mazzucchelli, "The customer should not necessarily care about the software language or how the program is written except as it applies to longevity. For example, if the program is written in a proprietary language for one brand of computer, be skeptical. You might want to move that software system in the future to another kind of computer."

Most important, open communi-

cation, typically called customer support, between the software vendor and user must exist. This communication runs both ways: Not only should users be able tell vendors of bugs, but if the vendor finds or hears of a bug, some mechanism should exist for vendors to tell users about that bug and when it will be corrected. For instance, ASK has a users conference and customer surveys, and updates its customers about bugs.

However, as with purchasing any other product or service, the user should investigate the vendor's reputation. The customer should visit customer references; see what their application is, how the software is used, and its benefits. Adds Williams, "Get an objective assessment as to how well the product is holding up and how well the vendor supports it. Then deal with the vendor." MA

SOFTWARE QUALITY ACTION BOX

	CIRCLE NUMBER
ASK Computer Systems Inc. <i>Mountain View, Calif.</i>	51
American Society for Quality Control <i>Milwaukee, Wis.</i>	52
Cadre Technologies Inc. <i>Providence, R.I.</i>	53
Comac Systems Corp. <i>Framingham, Mass.</i>	54
HEI Corp. <i>Carol Stream, Ill.</i>	55
IEEE Computer Society <i>Washington, D.C.</i>	56
ITP Enterprises Software Inc. <i>Cambridge, Mass.</i>	57
TravTech Inc. <i>Hartford, Conn.</i>	58

Chart: PIX/Elation/Fran Heyl Assoc., NYC. Source: Communications of the ACM